



Prérequis

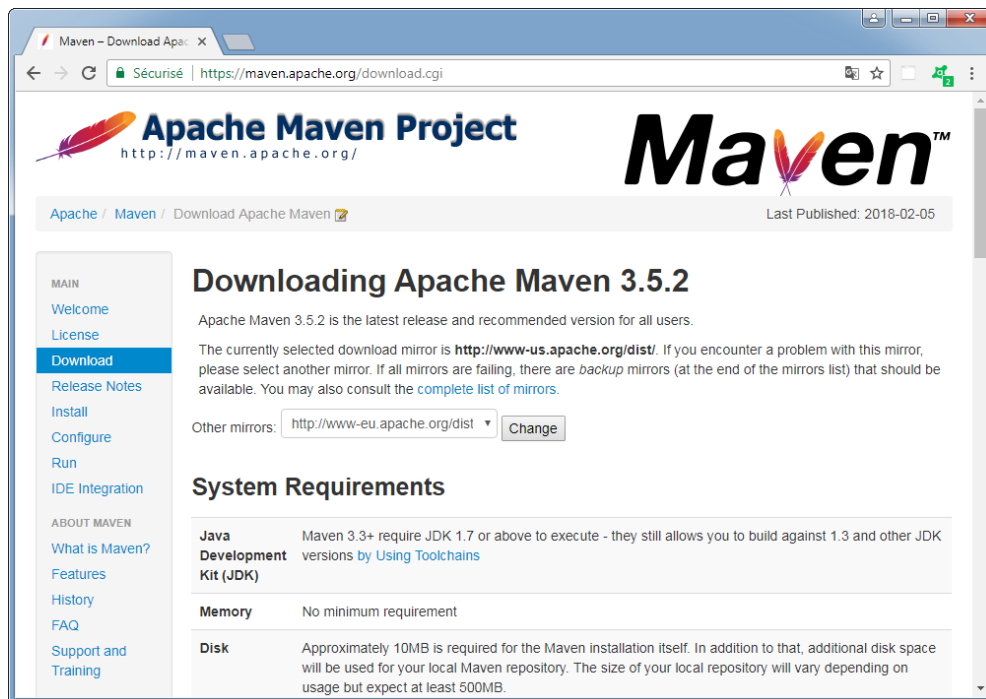
- Java : JDK 1.7 ou plus récent.
- RAM : aucun prérequis.
 - 1 Go de RAM libre est conseillé pour des projets de taille moyenne.
- Espace disque : aucun prérequis.
- Système d'exploitation : Aucun prérequis.
 - Disponible sous Windows, Linux, MAC, etc...

Installation de l'outil Maven

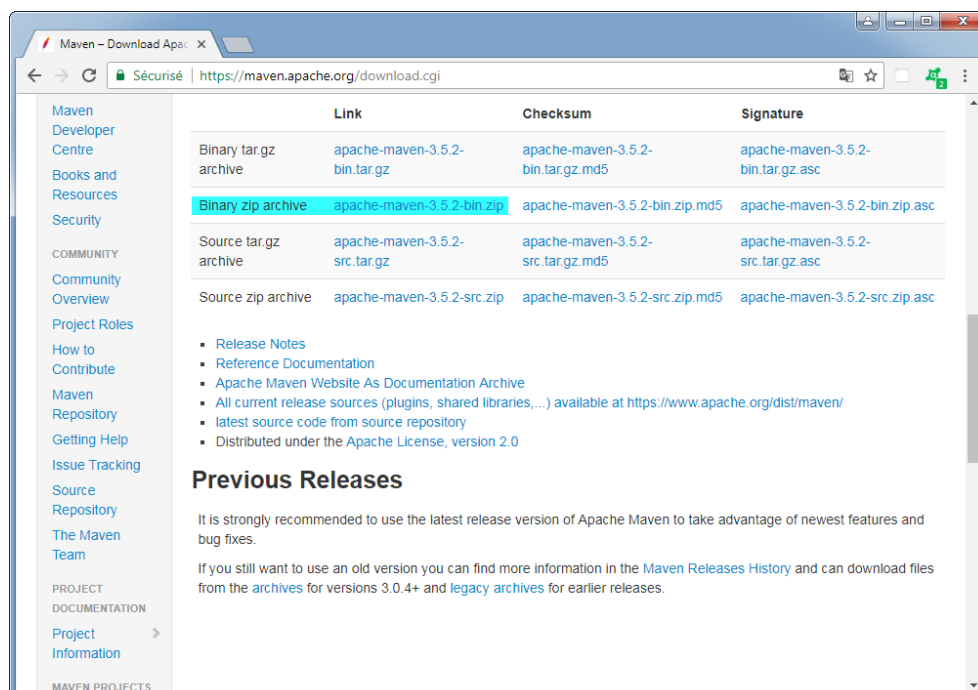
I. Sous Windows

Etape 1: Téléchargement de Maven

- Télécharger Maven depuis son site web officiel <https://maven.apache.org/download.cgi/>



- Choisir une version adaptée au système d'exploitation hôte.



Etape 2: Installation de Maven

- Ouvrez l'invite de commande. À partir de l'invite de commande, accédez au répertoire où le fichier apache-maven-x.x.x-bin.zip est enregistré et décompressez l'archive sous le chemin :

```
C:\integ_continue\maven\apache-maven-x.x.x
```

- Exécutez les commandes suivantes pour paramétrer les variables d'environnement requises pour le fonctionnement de Maven:

- Définir la variable d'environnement JAVA_HOME :

```
SET JAVA_HOME="C:\Program Files\Java\jdk1.8.0_xxx"
```

- Définir la variable d'environnement M2_HOME :

```
SET M2_HOME=C:\integ_continue\maven\apache-maven-x.x.x
```

- Définir la variable d'environnement M2 :

```
SET M2=%M2_HOME%\bin
```

- Ajouter les commandes Maven dans le PATH du système d'exploitation :

```
SET PATH=%PATH%;%M2%
```

```
C:\Windows\system32\cmd.exe

c:\integ_continue\maven>SET JAVA_HOME="C:\Program Files\Java\jdk1.8.0_161"
c:\integ_continue\maven>SET JAVA_HOME
JAVA_HOME="C:\Program Files\Java\jdk1.8.0_161"
c:\integ_continue\maven>SET M2_HOME=C:\integ_continue\maven\apache-maven-3.5.2
c:\integ_continue\maven>SET M2_HOME
M2_HOME=C:\integ_continue\maven\apache-maven-3.5.2
c:\integ_continue\maven>SET M2=%M2_HOME%\bin
c:\integ_continue\maven>SET M2
M2=C:\integ_continue\maven\apache-maven-3.5.2\bin
M2_HOME=C:\integ_continue\maven\apache-maven-3.5.2
c:\integ_continue\maven>
```

- Testez l'installation de Maven en exécutant la commande:

```
mvn -version
```

```
C:\Windows\system32\cmd.exe

c:\integ_continue\maven>mvn -version
Apache Maven 3.5.2 (138edd61fd100ec658bfa2d307c43b76940a5d7d; 2017-10-18T09:58:13+02:00)
Maven home: C:\integ_continue\maven\apache-maven-3.5.2\bin\..
Java version: 1.8.0_161, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_161\jre
Default locale: fr_FR, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
c:\integ_continue\maven>
```

- Exécutez la commande suivante pour afficher l'aide Maven:

```
mvn -help
```

II. Sous Linux (Centos 7)

Etape 1 : Installation du Java

1. Nous utiliserons open java pour notre démo, Obtenez la dernière version de <http://openjdk.java.net/install/>

```
yum install java-1.8*  
#yum -y install java-1.8.0-openjdk
```

2. Configuration de la variable d'environnement JAVA_HOME

1. Déterminez le bon emplacement de la version JAVA8.

```
find /usr/lib/jvm/java-1.8* | head -n 3
```

2. Créez un fichier appelé java.sh dans le répertoire /etc/profile.d/

```
vi /etc/profile.d/java.sh
```

3. Ajoutez le contenu suivant

```
#!/bin/bash  
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.282.b08-1.e17_9.x86_64  
export PATH=$PATH:$JAVA_HOME
```

NB: vous devez changer le chemin du java /usr/lib/jvm/java-1.8.0-**x86_64. Par le chemin généré dans la commande précédente.**

4. Enregistrez et fermez le fichier. Rendez-le exécutable à l'aide de la commande suivante.

```
chmod +x /etc/profile.d/java.sh
```

5. Ensuite, définissez les variables d'environnement de manière permanente en exécutant la commande suivante :

```
source /etc/profile.d/java.sh
```

6. Maintenant, vérifiez la version de fourni en utilisant la commande :

```
java -version
```

Etape 2 : Installation du Maven

- Télécharger Maven depuis son site web officiel <https://maven.apache.org/download.cgi/>
- Choisir et télécharger une version adaptée au système d'exploitation hôte.

```
# wget https://downloads.apache.org/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz
```

- Extraire l'archive de distribution dans le dossier /opt

```
tar xzvf apache-maven-* -C /opt
```

- Créez un fichier appelé maven.sh dans le répertoire /etc/profile.d/

```
vi /etc/profile.d/maven.sh
```

- Ajoutez le contenu suivant

```
#!/bin/bash

export M2_HOME=/opt/apache-maven-3.6.3
export M2=$M2_HOME/bin
export PATH=$PATH:$M2
```

NB: vous devez changer le chemin de l'outil maven par celui installé dans votre instance.

- Enregistrez et fermez le fichier. Rendez-le exécutable à l'aide de la commande suivante.

```
chmod +x /etc/profile.d/maven.sh
```

- Ensuite, définissez les variables d'environnement de manière permanente en exécutant la commande suivante :

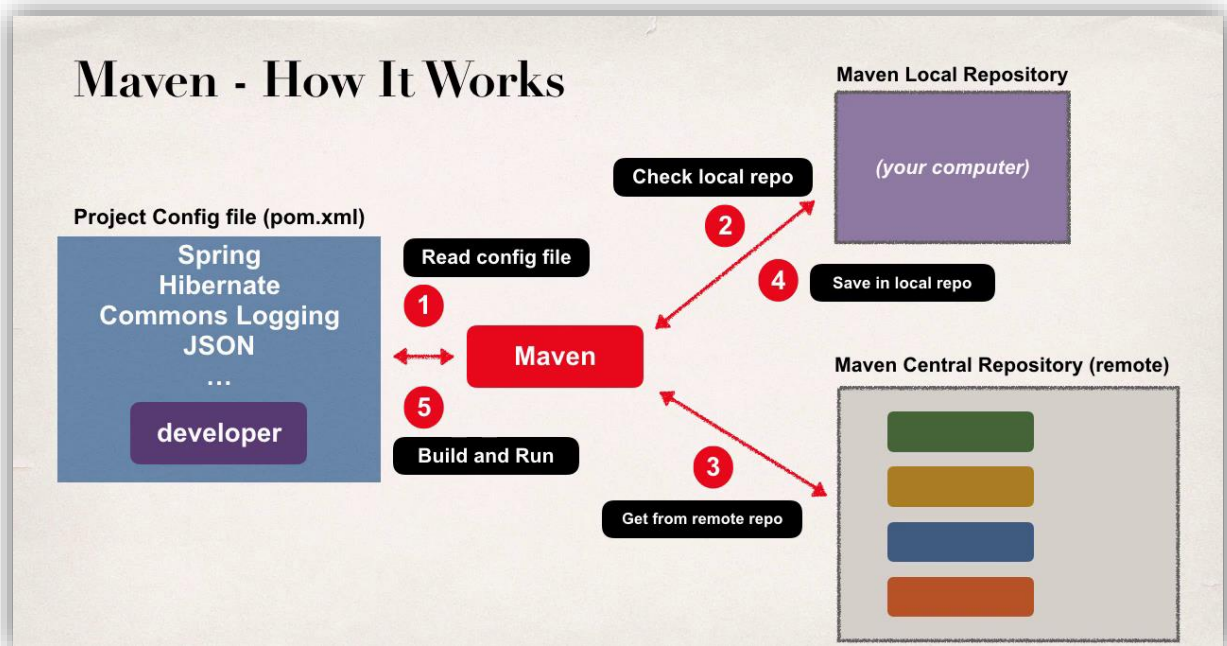
```
source /etc/profile.d/maven.sh
```

- Maintenant, vérifiez la version de fourni en utilisant la commande :

```
mvn -version
```

III. Génération d'un projet Maven

Maven repository



Project Object Model « pom.xml »

POM signifie "Project Object Model". Il s'agit d'une représentation XML d'un projet Maven contenu dans un fichier nommé pom.xml.

Exemple

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<packaging>jar</packaging>

<groupId>org.codehaus.mojo</groupId>
<artifactId>my-project</artifactId>
<version>1.0</version>

<!-- Build Settings -->
<build>...</build>
<reporting>...</reporting>
</project>
```

Générer une Application java de type Maven

- Générer l'architecture du projet

```
mvn archetype:generate
```

- Choisir le code par défaut (1839)

```
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1839:
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
7: 1.3
8: 1.4
Choose a number: 8:
Define value for property 'groupId': org.gk.cusrsusdevops
Define value for property 'artifactId': AppJavaMaven
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.gk.cusrsusdevops: :
Confirm properties configuration:
groupId: org.gk.cusrsusdevops
artifactId: AppJavaMaven
version: 1.0-SNAPSHOT
package: org.gk.cusrsusdevops
Y: : y
```

- Vérifier l'architecture du projet

```
[root@worker-nodel ~]# tree AppJavaMaven/
AppJavaMaven/
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       ├── org
│   │       │   ├── gk
│   │       │   │   ├── cusrsusdevops
│   │       │   │   └── App.java
│   └── test
│       ├── java
│       │   ├── org
│       │   │   ├── gk
│       │   │   │   ├── cusrsusdevops
│       │   │   │   └── AppTest.java
```


- Compiler le projet

```
[root@worker-node1 AppJavaMaven]# mvn compile
```

- Créer un package

```
[root@worker-node1 AppJavaMaven]# mvn package
```

- Vérifier le résultat

```
[root@worker-node1 AppJavaMaven]# tree
.
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       ├── org
│   │       │   └── gk
│   │       │       ├── cusrsusdevops
│   │       │       └── App.java
│   └── test
│       ├── java
│       │   ├── org
│       │   │   └── gk
│       │   │       ├── cusrsusdevops
│       │   │       └── AppTest.java
└── target
    ├── AppJavaMaven-1.0-SNAPSHOT.jar
    ├── classes
    │   ├── org
    │   │   └── gk
    │   │       ├── cusrsusdevops
    │   │       └── App.class
    ├── generated-sources
    │   └── annotations
    ├── generated-test-sources
    │   └── test-annotations
    ├── maven-archiver
    │   └── pom.properties
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   ├── default-compile
    │       │   │   ├── createdFiles.lst
    │       │   │   └── inputFiles.lst
    │       └── testCompile
    │           ├── default-testCompile
    │           │   ├── createdFiles.lst
    │           │   └── inputFiles.lst
    └── surefire-reports
        ├── org.gk.cusrsusdevops.AppTest.txt
        └── TEST-org.gk.cusrsusdevops.AppTest.xml
```

```

└─ test-classes
  └─ org
    └─ gk
      └─ cusrsusdevops
        └─ AppTest.class

```

32 directories, 13 files

- Tester le projet

```

[root@worker-node1 AppJavaMaven]# java -cp target/AppJavaMaven-1.0-
SNAPSHOT.jar org.gk.cusrsusdevops.App
Hello World!

```

- Créer la documentation de l'application

```

[root@worker-node1 AppJavaMaven]# mvn site
[root@worker-node1 AppJavaMaven]# tree target/site/
target/site/
├─ css
│   ├── maven-base.css
│   ├── maven-theme.css
│   ├── print.css
│   └─ site.css
├─ dependencies.html
├─ dependency-info.html
├─ images
│   ├── close.gif
│   ├── collapsed.gif
│   ├── expanded.gif
│   ├── external.png
│   ├── icon_error_sml.gif
│   ├── icon_info_sml.gif
│   ├── icon_success_sml.gif
│   ├── icon_warning_sml.gif
│   ├── logos
│   │   ├── build-by-maven-black.png
│   │   ├── build-by-maven-white.png
│   │   └─ maven-feather.png
│   └─ newwindow.png
├─ index.html
├─ plugin-management.html
├─ plugins.html
├─ project-info.html
└─ summary.html

```

- Nettoyer le projet

```
[root@worker-node1 AppJavaMaven]# mvn clean
```

- Testez la construction du projet `AppJavaMaven` en exécutant la commande

```
[root@worker-node1 AppJavaMaven]# mvn clean install
```

Gérer les dépendances d'un projet automatiquement avec Maven

- Modifier le code source de classe java principale **App.java**

```
package org.gk.cusrsusdevops;
import org.slf4j.*;
public class App{
    public static void main( String[] args ){
        System.out.println( "Hello World!" );
        Logger logger = LoggerFactory.getLogger(App.class);
        logger.info("Hello world");
    }
}
```

- Compiler le projet et analyser le résultat
- Ajouter les dépendances du package `slf4j` (<https://mvnrepository.com/>)

```
...
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
  <version>1.7.30</version>
</dependency>
...
```

- Compiler à nouveau le projet et analyser le résultat

Gérer les plugins d'un projet automatiquement avec Maven

Le plugin « `maven-compiler-plugin` » est ajouté automatiquement lors de la génération du projet et utilisé par Maven pour compiler les projets.

Si n'est pas spécifier, le projet sera compilé par la dernière version disponible dans le serveur.

- Modifier la configuration du Plugin

```
...
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.8.0</version>
  <configuration>
    <source>1.4</source>
    <target>1.4</target>
  </configuration>
</plugin>
```

```

    </configuration>
  </plugin>
...

```

- Ajouter une classe de test ListeVoitures.java

```

package org.gk.cusrsusdevops;
import java.util.*;
public class ListeVoitures
{
    public static void main( String[] args )
    {
        List<String> listVoitures = new ArrayList<String>();
        listVoitures.add("Peugeot");
        listVoitures.add("Citroen");
    }
}

```

- Compiler le projet et analyser le résultat
- Modifier la configuration du Plugin pour le changer à la version 1.5
- Compiler à nouveau le projet et analyser le résultat
- Supprimer le plugin «maven-compiler-plugin» puis Compiler à nouveau le projet et analyser le résultat

Générer une Application Web

- Générer l'architecture du projet webapp (1844)

```

mvn archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes
-DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4

```

- Vérifier l'architecture du projet

```

[root@worker-node1 ~]# tree webapp/
webapp/
├── pom.xml
├── src
│   └── main
│       └── webapp
│           ├── index.jsp
│           ├── WEB-INF
│           └── web.xml
4 directories, 3 files

```

- Ajout du plugin Jetty pour tester rapidement l'application web créée. Modifier le fichier pom.xml et ajouter la définition du plugin

```

...
<plugin>

```

```
<groupId>org.mortbay.jetty</groupId>  
<artifactId>maven-jetty-plugin</artifactId>  
<version>6.1.26</version>  
</plugin>  
...
```

- Exécuter le plugin Jetty

```
mvn jetty:run
```

- Tester l'application à partir d'un navigateur web sur le port 8080
- Modifier le code de la page index.jsp et vérifier les mises à jour

Apache Ant

Installation de Apache Ant

- Téléchargez la dernière version depuis la page de téléchargement du projet : <https://ant.apache.org/bindownload.cgi>

```
wget https://downloads.apache.org//ant/binaries/apache-ant-1.10.9-bin.tar.gz
```

- Extrayez-le dans le dossier /opt en utilisant la commande

```
tar xvfz apache-ant-1.10.9-bin.tar.gz -C /opt/
```

- Créez un lien symbolique vers le dossier vers le binaire ant pour qu'il soit accessible par tous les utilisateurs.

```
ln -s /opt/apache-ant-1.10.9/bin/ant /usr/local/bin/
```

Configuration de la variable d'environnement ANT

- Créez un fichier appelé ant.sh dans le répertoire /etc/profile.d/

```
vi /etc/profile.d/ant.sh
```

- Ajoutez le contenu suivant :

```
#!/bin/bash
ANT_HOME=/opt/apache-ant-1.10.9
PATH=$ANT_HOME/bin:$PATH
export PATH ANT_HOME
export CLASSPATH=.
```

- Enregistrez et fermez le fichier. Rendez-le exécutable à l'aide de la commande suivante.

```
chmod +x /etc/profile.d/ant.sh
```

- Ensuite, définissez les variables d'environnement de manière permanente en exécutant la commande suivante :

```
source /etc/profile.d/ant.sh
```

- Maintenant, vérifiez la version de fourni en utilisant la commande :

```
ant -version
```

Exemples d'utilisation de « ant »

Exemple 1

- Créez le fichier build.xml suivant :

```
<project name="MyProject" default="main">
  <target name="main">
    <echo message="Hello World !"/>
  </target>
</project>
```

- Lancez la commande suivante pour démarrer le build

```
[root@worker-node1 ant]# ant -f build.xml
```

Exemple 2

- Créer la classe HelloWorld.java avec le contenu suivant

```
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
```

- Créez le fichier de configuration build.xml suivant

```
<?xml version="1.0" ?>
<project name="Hello World" default="compress">

    <target name="compile">
        <javac srcdir="."/>
    </target>
</project>
```

```
<echo> Compilation Complete! </echo>
</target>

<target name="compress" depends="compile">
    <jar destfile="HelloWorld.jar" basedir="." includes="*.class" />
    <echo> Building .jar file Complete! </echo>
</target>

</project>
```

- Lancez la commande suivante pour compiler le programme

```
[root@worker-node1 Example2]# ant compile
```

- Lancez la commande suivante pour créer l'archive

```
[root@worker-node1 Example2]# ant compress
```

- Supprimez les fichiers HelloWorld.class HelloWorld.jar. Puis lancez la commande suivante pour démarrer le build complet de votre programme

```
[root@worker-node1 Example2]# ant -f build.xml
```

- Testez votre classe

```
[root@worker-node1 Example2]# java -cp HelloWorld.jar HelloWorld
Hello World!
```

Exemple 3

- Reprenez l'exemple 2 et modifiez le fichier build.xml avec le contenu suivant

```
<?xml version="1.0" ?>
<project name="HelloWorld" default="compress">

    <target name="init">
        <mkdir dir="build/classes" />
        <mkdir dir="dist" />
    </target>

    <target name="compile" depends="init">
        <javac srcdir="." destdir="build/classes" />
    </target>
```



```
<target name="compress" depends="compile">
    <jar destfile="dist/HelloWorld.jar" basedir="build/classes" />
</target>

<target name="execute" depends="compile">
    <java classname="HelloWorld" classpath="build/classes" />
</target>

<target name="clean">
    <delete dir="build" />
    <delete dir="dist" />
</target>

</project>
```

- Essayez les cibles suivantes :
 - **init**
 - **compile**
 - **compress**
 - **execute**
- Testez votre classe
- Nettoyez la trace de construction du projet

Composer

Composer est un outil de gestion de dépendance en PHP. Il vous permet de déclarer les librairies dont votre projet dépend, et il va vous les gérer (installation/mise à jour). C'est vraiment l'outil de gestion de dépendance en PHP.

Installer composer

- Commencez par installer le logiciel de prise en charge

```
yum install php-cli php-zip php-xml wget unzip
```

- Téléchargez le script d'installation de Composer

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
```

- Vérifiez l'intégrité du téléchargement

```
HASH="$(wget -q -O - https://composer.github.io/installer.sig)"
```

- Utilisez le script suivant pour comparer le hachage officiel à celui que vous avez téléchargé:

```
php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"
```

- Installez Composer

```
php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

- Lorsque le programme d'installation a terminé le processus, vérifiez s'il fonctionne correctement :

```
composer
```

➔

```
[root@worker-node2 composer]# composer
```



```
Composer version 2.0.8 2020-12-03 17:20:38
```

```
Usage:
```

```
command [options] [arguments]
```

Exemple

- Créez un répertoire racine de projet pour un fichier avec la description du projet, ses dépendances et d'autres informations supplémentaires

```
mkdir ~/c_sample
```

- à l'aide de l'outil composer, charger un package, exemple : monolog/monolog. Le site Web packagist.org propose une large gamme de packages PHP.

```
composer require monolog/monolog
```

- vérifiez si tous les fichiers ont été créés en listant le contenu du répertoire, puis affichez le contenu du fichier composer.json

```
ls -l  
cat composer.json
```

- Créez un fichier de test « composer_sample.php » et y ajoutez le contenu suivant :

```
<?php  
  
require __DIR__ . '/vendor/autoload.php';  
use Monolog\Logger;  
use Monolog\Handler\StreamHandler;  
  
// create a log channel  
$log = new Logger('name');  
$log->pushHandler(new StreamHandler('/~/c_sample/text.log',  
Logger::WARNING));
```

```
// add records to the log
$log->warning('Foo');
$log->error('Bar');
```

- utiliser la commande suivante pour charger automatiquement monolog :

```
php composer_sample.php
```

- Vérifiez le résultat de l'exécution dans le fichier `~/c_sample/text.log`

```
[root@worker-node2 composer]# cat text.log
[2021-01-17 17:31:10] name.WARNING: Foo [] []
[2021-01-17 17:31:10] name.ERROR: Bar [] []
[2021-01-17 17:32:16] name.WARNING: Foo [] []
[2021-01-17 17:32:16] name.ERROR: Bar [] []
[2021-01-17 18:06:31] name.WARNING: Foo [] []
[2021-01-17 18:06:31] name.ERROR: Bar [] []
```